

times, slacks, and capacitances at any node in the network.

Use of GAFG began in earnest beginning with the 620 project. It works on an in-core gate level model comprising a collection of macros. In practice, this collection of macros is either an entire chip, or one of the functional units of a chip (e.g., the floating point unit), depending on the model size. This contrasts with synthesis, which runs on only a single macro at a time

The process of performance tuning consists of 3 repeating steps: (see figure 3)

(1) the macros are synthesized using timing constraints generated in the previous iteration.

(2) the flat chip or unit netlist is built by flattening the macros and other associated logic, and is timed using STEP.

(3) the same netlist is used to generate new timing constraints for the macros.

The timing constraints must be manually generated for the first iteration, in order to “seed” the process. As the process is repeated, the chip’s worst case cycle time will improve to a point, then hold steady. At this point, if the cycle time is still unacceptable, the tuning loop is interrupted to edit the DSL/1 source, and then resumed. This is expected, since the performance of the logic generated by synthesis is not independent of how the DSL/1 is coded.

4.2 GAFG Operation

GAFG can see what synthesis can’t: the effect on timing that a change in one macro will have on other macros. To actually do logic transformations at the chip level would be too costly in time and memory, due to the size of the netlist. Therefore, GAFG simply posits a set of reasonable timing changes by scaling the delays with a scale factor. These changes to the network should help to meet the target cycle time, and are applied to the timing model without actually changing the logic. Each time GAFG applies a set of changes, the model is retimed. After retiming, the changes are refined based on the resultant slacks, and the process is repeated. After the last retiming of the network, the arrival times at the boundary of each macro in the model are written to a file, to be used as the timing constraints for the next pass of synthesis.

To ensure that the posited changes are reasonable, only logic that was synthesized is allowed to be “changed”. Also, the amount of delay improvement allowed for a given path is limited to a percentage of its original delay, set by the designer.

Figure 5. Fraction of the nets which meet the timing constraints during the design process.

5. Conclusions

In figure 4 we can see how the worst slack in the 620 gradually decreased over the duration of the project. The dashed line uses a delay estimate without detailed RC models for the nets.

Despite the adverse effects of the flush mode cycle stealing algorithm on constraint generation, STEP has been successful in the timing verification of the PowerPC chip set, offering high performance and broad analysis capabilities. Also, as seen in figure 4 and figure 5, the GAFG/synthesis loop contributes greatly to cycle time reduction.

6. Acknowledgments

Many people were involved in this project. We would like to thank Krishna Belkhale, Thomas Burnett, Shervin Hojat, Stephen Lim, Joe Rahmeh, Leon Stok, Bob Swanson, Louise Trevillyan and J.P. Watson.

7. References

- [1] Brodnax, T. and M. Schiffli, F. Watson: “The PowerPC 601 Design Methodology”, in Proc. Int. Conf. Computer Design, pp. 248-252. Cambridge, Mass., Oct. 3-6, 1993
- [2] Roth, C. and T. Brodnax: “The PowerPC 604 Design Methodology”, in Proc. Int. Conf. Computer Design, Cambridge, Mass. Oct. 10-12, 1994
- [3] Qian, J. and S. Pulella, L. T. Pillage: “Modelling the Effective Capacitance of RC Interconnect”, IEEE Transactions on Computer-Aided-Design, May 1993.
- [4] Ratzlaff, C.L. and L.T. Pillage: “RICE: Rapid Interconnect Circuit Evaluation using AWE”, Proceedings of the Custom Integrated Circuits Conference, May 1992.
- [5] Szymanski, T.G. and N. Shenoy: “Verifying Clock Schedules”, IEEE Design Automation Conference, 1992.
- [6] Tsay Ren-Song and I. Lin: “Robin Hood: A System Timing Verifier for Multi-Phase Level Sensitive Clock Designs”, IBM Technical Report 17272, IBM Corporation, 1991.
- [7] van Ginneken, L.P.P.P., “Fanin Ordering in Multi-Slot Timing Analysis”, in Proc. Int. Conf. Computer Design, Cambridge, Mass., Oct. 11-14, 1992

Figure 4. 620 Performance tuning slack improvement

prefer the more graceful failure characteristics of automatic test pattern generation over the all-or-nothing failure behavior of BDDs.

The connection count reduction phase is then followed by a level reduction phase in which synthesis attempts to reduce the number of logic levels. Pattern matching and covering algorithms are used to do the technology mapping

3.2 Integrated Timing Analysis

To perform good optimization in the first place, it is necessary to be able to evaluate the object function accurately. In timing this is fundamentally more difficult than in area optimization, since in area optimization the object function is cumulative: any local change in the area translates to an equal change in the global area. In timing however, the situation is not so simple. A local change in delay does not necessarily have an effect on the slack of the critical path if it is not on the critical path, or if there are multiple critical paths. Furthermore, even in a simple delay model, delay is not a property of a single cell. The delay of a cell can only be computed in context, by evaluating the delay equation.

Because of these considerations it was necessary to use a full-featured static timing analyzer to evaluate any proposed changes. The static timing analyzer is tightly integrated with the synthesis algorithms and uses the same in core model. The timing analyzer is accessed through an API. The timing analyzer is incremental, demand driven, and satisfies slack queries through lazy evaluation.

IBM has developed several such timing analyzers, which are all compatible, illustrating their importance to the design of high performance processors. The 601, 603 and 604 chips used SlackHoe as the synthesis timer. 620 used EinsTimer. A third synthesis timer, named Vampire, is still under development.

3.3 Timing Optimization Algorithms

The transforms for timing correction are based on greedy heuristics. It should be noted that this is true for almost every logic synthesis algorithm ever proposed. Algorithms generally differ in the kinds of incremental changes they allow, the sophistication with which correctness is established and the accuracy with which the object function is predicted. For instance, in transduction, the proposed changes are simple. Picking the functionally correct ones takes a lot of sophistication. The timing optimization algorithms in logic synthesis are not sophisticated with respect to proving correctness of the changes or with respect to area, but they are accurate with respect to their objective function: slack.

With the exception of fanin ordering algorithms [7], no timing optimization algorithms are known which can find an optimal solution without explicit enumeration of a large number of possible solutions. It is necessary to change the logic network before the change can be evaluated by the integrated timing analyzer. Therefore most timing optimization transforms are written such that a proposed change is first made and then undone. Typically a large number of changes must be attempted before a change can be found that improves the timing.

The standard cell libraries used for most of the PowerPC chips have a large number of implementations of the same gate, which differ in drive capability, which we will call *power levels*. To get reasonable timing estimates it is necessary to select good power levels. This is so important that the effect of a change can only be evaluated if the power levels of all affected gates have been readjusted. Throughout timing optimization, power levels are locally re-optimized for every change. The best power level is selected by evaluating all power levels.

The timing optimization starts after technology mapping with a network which is (mostly) optimized for area. The timing optimization transforms are guided by the slack returned by the timing analyzer. On paths which have negative slack, the slacks of various paths are traded off against each other, such that the worst slack

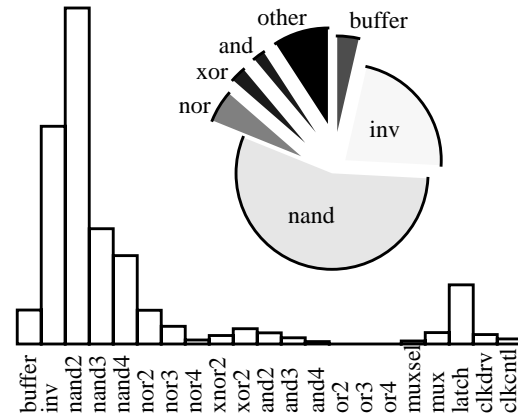


Figure 2. The relative frequency of standard cells used in the 603.

gets better, possibly at the expense of the slack of another path. This type of optimization is especially applied a lot to the critical path to increase the overall critical slack. This type of optimization tends to make a lot of paths approximately the same length. Transforms that are applied are buffer insertion, parallel copying of standard cells, power level optimization, fanin ordering, inverter optimizations, re-mapping, expansion of complex cells and others. As can be seen from figure 2, for performance reasons mostly small NAND gates were used.

When the negative slacks have been eliminated, or when no further progress can be made, area optimization is attempted on the positive slack paths. The timing optimization also attempts to satisfy fanout and transition time constraints. For each standard cell there is a limit on the number of sinks and the capacitance it can drive, as well as a maximum transition time for each input.

4. Timing Constraints Generation

4.1 Role of Global Assertion File Generation in Timing Verification and Optimization

Timing constraints for each macro are generated automatically by GAFG (Global Assertion File Generation, pronounced "gaf-gee"). These constraints are used by the SlackHoe or EinsTimer timers during logic synthesis. The constraints are generated such that the slack on paths that traverse more than one macro is realistically apportioned among those macros.

GAFG is tightly integrated with STEP, which it accesses through an API. Thus, it can easily set or query delays, arrival

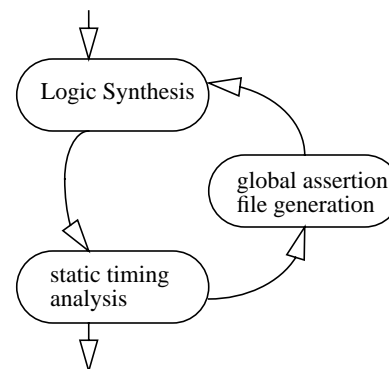


Figure 3. The design iteration loop using global assertion file generation

either the clock or data signal through the latch, both signals needed to be in the same time zone.

The logic topology resulted in the development of an edge sensitive phase tag algorithm which allowed for the distinction of the clocked origination of all data signals. Using this tagging method, we were able to accurately calculate slacks for both full and half cycle paths with no designer intervention necessary to manually adjust the clock to the correct cycle for the test of the data. We also developed a technique to automatically relaunch transparent latch data phases in the correct time zone, without wandering into the next machine cycle, or requiring manual data adjusts to the data arrival times to insure correct signal propagation.

2.5 Dynamic Logic Analysis

Dynamic CMOS gates were timed by denoting for each input/output pin path pair whether the path propagated only the rising or falling transition, or if the path was clocked precharge or evaluate. For a given evaluation of a dynamic gate, only a rising or falling transition was propagated, not both. The rise and fall transitions of the clock feeding the dynamic circuit were used to launch precharge-and-evaluate times with distinct tags. We further provided clock consistency checks between dynamic circuits with intervening logic which guarantee that a stream of dynamic domino logic is precharged and evaluated with the same clock edges. Hold and setup checks were also performed for the dynamic logic by the timing analyzer.

2.6 Electrical Analysis

The effects of gate wire interconnection are imported via an RC delay file and a net capacitance file. These files are created via a tool known as Hercules which extracts the net for analysis from the physical model.

The RC delay is determined by deriving a reduced order transfer model for RLC circuits using the Rapid Interconnect Circuit Evaluator [4]. RICE uses the AWE (Asymptotic Waveform Evaluation) technique for its analysis. The RC delay file contains the poles and residues representative of the circuit model of the extracted net. The poles and residues along with the transition time of the gate driving the net are passed to RICE to calculate the interconnect delay and the transition time of the signal at the sink point. This yields much more accurate wire delays than older techniques where the RC delay was simply added to the gate delay.

STEP also incorporates the $C_{\text{effective}}$ capacitance model[3] to yield less pessimistic delays for gates where the resistive shielding of the net causes a reduced apparent capacitance to be visible to the output stage of the gate. The model is incorporated by deriving the moments of the extracted net from the physical data by Hercules to represent the effective capacitance. This information is then passed along with the input transition time to the gate signal to RICE to compute the effective delay of the gate based upon the driving point admittance model. This technique has shown noted delay improvement in a number of the critical paths.

2.7 Results Analysis

STEP provides the designer with a number of different reporting options, detailing path slacks and book delays. In addition, a graphical tool known as Pirate is used to interactively view timing results, traverse the logic model, and provide a method for making minor logic changes and reinvoking the timer to see the effect of the model changes on the timing.

2.8 Performance Comparisons

As can be seen from table 1, STEP is a high performance timing analyzer, capable of providing full path slack analysis to the logic designer in a relatively short period of time. This performance is critical to the design tuning process, and to such applications as the Global Assertion File Generator.

Chip	I/O Pins	Run Time	Timer Iterations
601	149911	4m 38s	1
603	190525	6m 7s	1
604	281481	12m 48s	Cycle Stealing: 4
620	599999	39m 18s	Cycle Stealing: 5

Table 1: STEP Performance Comparisons

2.9 Experience Notes

One of the main difficulties experienced was the effect of negative slack paths on topological cycle stealing loops, and the resultant exit arrival times calculated by the timer at the convergent solution. Cycle stealing paths with more delay than allowable by the number of latching elements in the loop amplify the signal arrival times during the process of achieving arrival time convergence at the loop breakpoints. Latch launch times in the loop converge at the time of the closing clock edge constraint, not necessarily representative of a path delay through the loop. The false path delay times that result caused problems for GAFG, making it difficult to determine how and which paths should have timing correction applied. Furthermore, as the design's failing slack approached but was still less than zero, many iterations of the timer were required to achieve a convergent solution since the arrival time creepage at the loop breakpoints was small. In summary, while the flush mode algorithm guarantees correct timing operation of the circuit, it hinders the constraint generation needed to achieve timing optimization.

3. Timing Optimization in Logic Synthesis

3.1 Architecture of the Synthesis system

BooleDozer logic synthesis works on an in-core gate level model of the network. Initially the network is represented by means of abstract (technology independent) gates, for instance, nand gates of unlimited fanin and fanout. In the process of optimization, many changes are made to the logic, but no change in the level of abstraction is made other than the transition from abstract gates to technology mapped gates. The in-core model allows the simultaneous co-existence of technology mapped and abstract gates. The in-core model is manipulated by an Application Programming Interface (API). The synthesis algorithms are implemented as independent programs called *transforms* (figure 1).

Synthesis starts with technology independent optimizations, which aim at reducing the connection count. The techniques used here can mostly be found in the open literature, but it is worth mentioning that we do not use BDDs anywhere in synthesis. We

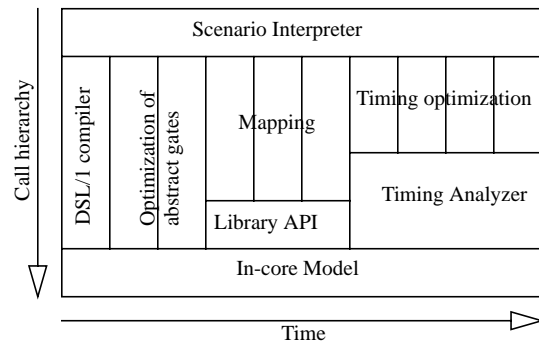


Figure 1. Global architecture of the BooleDozer synthesis system.

Timing Verification and Optimization for the PowerPC Processor Family

Robert E. Mains, Thomas A. Mosher
IBM RISC System/6000 Division
Austin, TX 78759

Lukas P.P.P. van Ginneken, Robert F. Damiano
IBM Research Division
Yorktown Heights, NY 10598

This paper presents the timing verification and optimization tools used for the 601, 603, 604 and 620 PowerPC¹ processor designs. The timing verification is done by static timing analysis at the chip level, while the timing optimization is done by synthesis at the macro level. A method for automatically deriving timing constraints for timing optimization is described.

1. Introduction

The PowerPC designs are described in DSL/1, an IBM proprietary Register Transfer Level design language. A DSL/1 design description is essentially a net list, written with the aid of high level constructs. The PowerPC designs are implemented using synthesized standard cells for control logic and more complex ‘Off-The-Shelf’ cells for data flow logic. For a description of the overall PowerPC development process refer to [1, 2].

Logic synthesis and timing optimization of the control logic was done by the BooleDozer² Synthesis system. It was originally developed for the 601 project, and was subsequently used on the 603, 604, 620 PowerPC processors, as well as other IBM products. Chip timing verification was performed by STEP, the Somerset Timing Evaluation Program. It is a high performance static timing analysis tool capable of analyzing the entire chip at the logic gate level. Macro timing constraints were generated by the Global Assertion File Generator (GAFG). These timing constraints are used in the next iteration of timing optimization. Both GAFG and STEP were developed at the Somerset Design Center.

The remainder of this paper is organized as follows: Section 2 describes the static timing analysis tool. Section 3 describes timing optimization, and section 4 describes timing constraint generation.

2. Timing Verification

2.1 The Timing Verification Problem

The logic topology and circuit design techniques used in the PowerPC presented a problem in the development of STEP. These designs are characterized by a single system master clock with a 50% duty cycle. The slave clock is approximately the inverse of the master clock. The 601 and 603 implementations of the PowerPC used only Master-Slave latch pairs. To achieve the performance objective the 604 and 620 processors used a disjoint latch design, in which the two latches can be separated by logic. Both latches were designed to be transparent, making cycle stealing possible, but also introducing troublesome loops into the timing analysis model.

The pervasive use of dynamic logic in the 620 design also exerted unique requirements on the development of the timing tool. Finally, more advanced methods were needed for the analysis of output capacitive loading and interconnect delays.

2.2 The Timer

Each logic gate’s input pin to output pin delay is represented by

a five coefficient delay equation of the form

$$(K_1 + K_2 \times T_x) \times C_l + K_3 \times C_l^2 + K_4 \times C_l + K_5$$

where T_x is the transition time of the input pin signal, C_l is the load capacitance driven by the output pin of the gate, and $K_1 - K_5$ are delay coefficients derived from a SPICE characterization of the circuit under designer specified operational conditions. Separate delay coefficients are used for computation of the rise and fall delays, as well as the rising and the falling transition times.

A forward delay calculation and arrival time propagation is performed first, followed by a reverse slack calculation and propagation. *Slack* is defined to be the required time minus the arrival time of the incoming data signal. The required time at an internal point is calculated from the required times at primary outputs or latches, by subtracting the delay of the paths to those points. At a primary output, the required time has to be specified as an assertion; at a latch the required time is derived from the closing edge of a latch clock plus a cycle adjust if necessary minus a setup constant characteristic of the latch.

2.3 Transparent Latch Analysis

The use of transparent latches necessitated the development of a cycle stealing algorithm for performing timing analysis. We implemented a combined derivative of algorithms proposed by [5, 6]. The method is sometimes referred to as the ‘flush mode’ cycle stealing algorithm, where late mode and early mode arrival times at latches are corrected based upon the timing constraints defined by the clock.

Arrival times are continually propagated until the logic model settles into a steady state. The steady state is determined by storing arrival times for each iteration of the delay calculator over the timing model at loop breakpoints. Loop breakpoints are determined by a timing graph cycle analysis, performed after a clock trace which breaks any cycles in the network. Legal loops are those in which latches are present, where a loop breakpoint is set on a latch data input pin. A topological ordering of pins is performed after all cycles in the timing graph have been removed. The topologically ordered list of pins is continuously processed by the timer until convergent solutions are reached at the loop breakpoints, or the iteration limit is exceeded. Note that we did not extract a specific latch graph for the analysis.

2.4 Half Cycle Path Analysis and Time Zone Changes

Static timing analyzers characteristically launch timing events within the clock cycle definition of the chip under analysis. Events are initiated by the leading edge of the slave clock, propagated through combinatorial logic, and captured by the trailing edge of the master clock. Timing events are defined within a single machine cycle of clocks, or a single *time zone*. The strict master-slave topology used in the 601 and 603 designs enforced proper launch of data signals at the beginning of the clock cycle with no extra work necessary by the timing analyzer. This was not true with the 604 and 620 designs, where each edge of the master clock initiated timing events. At a given latch the clock signal arrival time was in the base time zone, but the arriving data signal had a event time in the next time zone. To insure correct propagation of

1. PowerPC is a Registered Trademark of IBM Corp.
2. BooleDozer is a trademark of IBM Corp.